



# Floating pointer adder ISA

Prof. Alberto Borghese  
Dipartimento di Informatica  
[borgnese@di.unimi.it](mailto:borgnese@di.unimi.it)

Università degli Studi di Milano

Riferimenti sul Patterson, 5a Ed.: 3.4, 3.5, 4.2

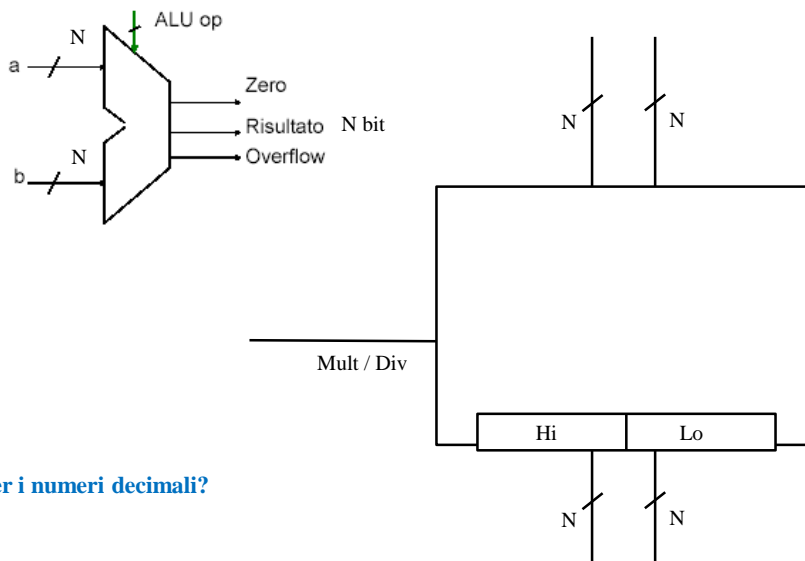


## Sommario

- **Somma in virgola mobile**
- ISA
- Istruzioni aritmetico-logiche



## Circuiti operazioni tra numeri interi



E per i numeri decimali?



## Codifica in virgola mobile Standard IEEE 754 (1980)

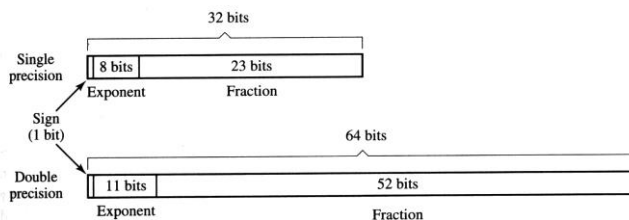


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Rappresentazione polarizzata dell'esponente:

Polarizzazione pari a 127 per singola precisione =>  
1 viene codificato come 1000 0000.

Polarizzazione pari a 1023 in doppia precisione.  
1 viene codificato come 1000 0000 000.



## Esempio di somma in virgola mobile



$$a = 7,999 \times 10^1 \quad b = 1,61 \times 10^{-1} \quad a + b = ?$$

NB I numeri decimali sono normalizzati.

Una possibilità è:

$$\begin{array}{r} 79,99 \quad + \\ 0,161 \quad = \\ \hline \end{array}$$

$$80,151 \times 10^0 = 80,151 \rightarrow 8,0151 \times 10^1 \text{ in forma normalizzata}$$

Altre possibilità sono:

$$\begin{array}{r} 799,9 \quad + \\ 1,61 \quad = \\ \hline \end{array}$$

$$\begin{array}{r} 7,999 \quad + \\ 0,0161 \quad = \\ \hline \end{array}$$

$$801,51 \times 10^{-1} = 8,0151 \times 10^1 \text{ in forma normalizzata}$$

$$8,0151 \times 10^1$$



## Quale forma conviene utilizzare?



Supponiamo di avere 4 cifre in tutto: 1 per la parte intera e 3 per la parte decimale

$$\begin{array}{r} 79,99 \quad + \\ 0,161 \quad = \\ \hline \end{array}$$

$$\begin{array}{r} 799,9 \quad + \\ 1,61 \quad = \\ \hline \end{array}$$

$$\begin{array}{r} 7,999 \quad + \\ 0,0161 \quad = \\ \hline \end{array}$$

$$80,151 \times 10^0$$

$$801,51 \times 10^{-1}$$

$$8,0151 \times 10^1$$

La rappresentazione **migliore** è:

$$\begin{array}{r} 7,999 \quad + \\ 0,0161 \quad = \\ \hline \end{array}$$

Risultato normalizzato

$$8,0151 \times 10^1$$

Con la quale posso scrivere: 1 cifra prima della virgola (8) e 3 cifre dopo la virgola (015), 1 va perso.

Con la rappresentazione più a sinistra, perdo le decine, con quella in mezzo decine e centinaia commettendo un errore grande sulla rappresentazione.

**Allineo al numero con esponente maggiore.**



# Approssimazione



Interi -> risultato esatto (o overflow)

Numeri decimali -> Spesso occorrono delle approssimazioni

- Troncamento (floor):  $1,001 \times 10^2$  (cf. Slide precedente)
- Arrotondamento alla cifra superiore (ceil):  $1,002 \times 10^2$
- Arrotondamento alla cifra più vicina:  $1,002 \times 10^2$

IEEE754 prevede 2 bit aggiuntivi nei calcoli per mantenere l'accuratezza.

bit di guardia (guard)

bit di arrotondamento (round)



## Esempio: aritmetica in floating point accurata



$$a = 2,34 \quad b = 0,0256$$

$$a + b = ?$$

Codifica su 3 cifre decimali totali.

Approssimazione mediante arrotondamento.

Senza cifre di arrotondamento devo scrivere:

$$2,34 +$$

$$0,02 =$$

-----

**2,36**

Con il bit di guardia e di arrotondamento posso scrivere:

$$2,3400 +$$

$$0,0256 =$$

-----

2,3656

L'arrotondamento finale (al più vicino) fornisce per rintrare in 3 cifre decimali fornisce: **2,37**



## L'effetto perverso del troncamento



$$C = A + B$$

if ( $C \leq A$ ) then

(a)...

else

(b)....

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = 8,0151$$

*Passando alla codifica su 4 bit ottengo:*

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = 8,015 \quad \Rightarrow \quad C > A \text{ correttamente}$$

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = 7,999161$$

*Passando alla codifica su 4 bit ottengo:*

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = 7,999 \quad \Rightarrow \quad C = A \text{ errore!!!}$$

Questo è un errore molto comune quando si considera l'aritmetica con i numeri decimali



## Non vale la proprietà associativa



$$Z = A + (B + C)$$

$$A = -10^{38} \quad B = 10^{38} \quad C = 1$$

$$(B + C) = 10^{38} \quad \Rightarrow \quad Z = 0$$

$$Z = (A + B) + C$$

$$(A + B) = 0 \quad \Rightarrow \quad Z = 1$$

Risultati molto diversi.



## Algoritmo di somma in virgola mobile - I



- 1) Trasformare **uno dei due numeri** in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola. Si allinea all'esponente più alto (denormalizzo il numero più piccolo).
- 2) Effettuare la somma delle mantisse.

Se il numero risultante è normalizzato termino qui. Altrimenti:

- 3) Normalizzare il risultato.



## Esempio di somma in virgola mobile - II



$$a = 9,999 \times 10^1 \quad b = 1,61 \times 10^{-1} \quad a + b = ?$$

Supponiamo di avere a disposizione 4 cifre per la mantissa e due per l'esponente.

Devo trasformare uno dei numeri, una possibilità è:

$$\begin{array}{r} 9,999 \times 10^1 + \\ 0,016 \times 10^1 = \quad \text{Perdo un bit perchè non rientra nella capacità della mantissa} \\ \hline 10,015 \times 10^1 \quad \text{Il risultato non è più normalizzato, anche se i due addendi sono normalizzati.} \end{array}$$

Normalizzazione per ottenere il risultato finale:

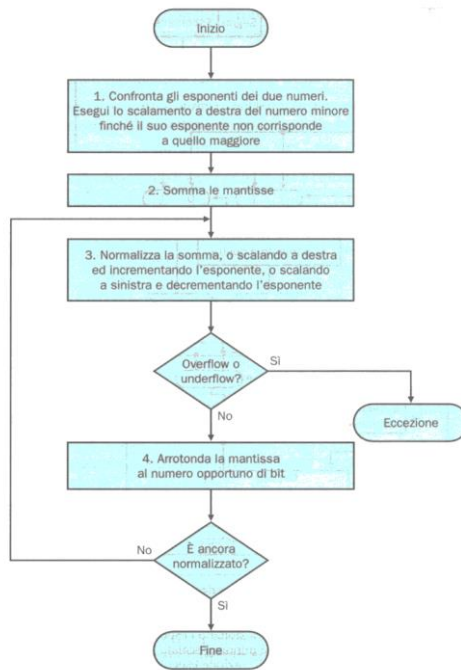
$$10,015 \times 10^1 = 1,001 \times 10^2 \text{ in forma normalizzata.}$$

NB: In questa fase si può generare la necessità di rinormalizzare il numero (passo 3):

$$\begin{array}{l} \text{Esempio: } 9,99999 \times 10^2 \Rightarrow \text{Arrotondo alla cifra più vicina} \Rightarrow 10,00 \times 10^3 \\ \quad \quad \quad \Rightarrow \text{Normalizzazione} \quad \quad \quad \Rightarrow 1,0 \times 10^4 \end{array}$$



# Algoritmo risultante

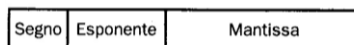


## Il circuito della somma floating point: gli attori



A + B

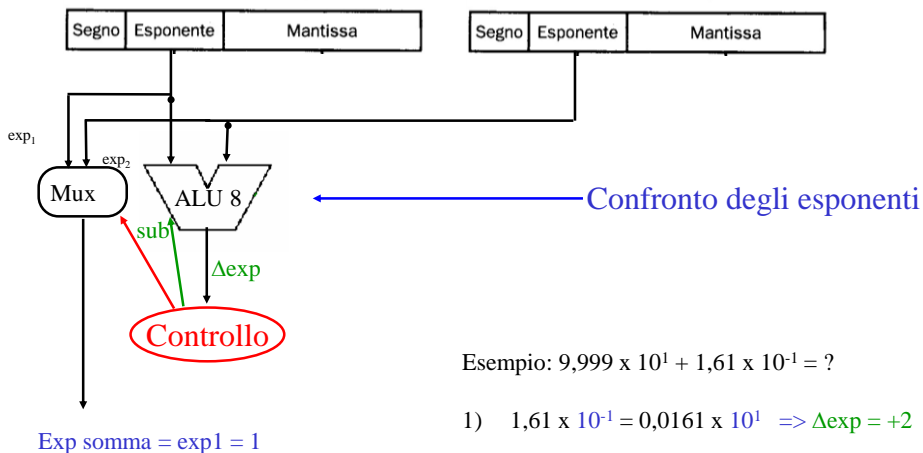
C



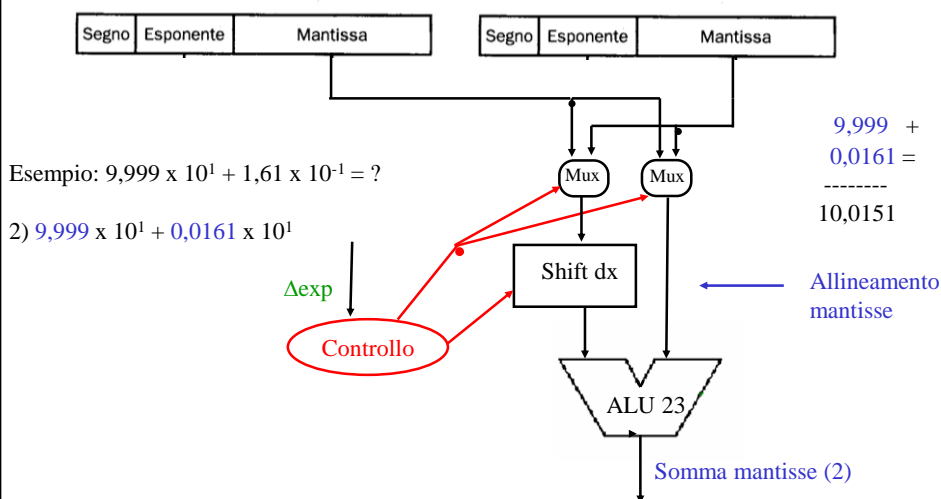
Rappresentazione normalizzata IEEE754



## Il circuito della somma floating point: determinazione dell'esponente comune



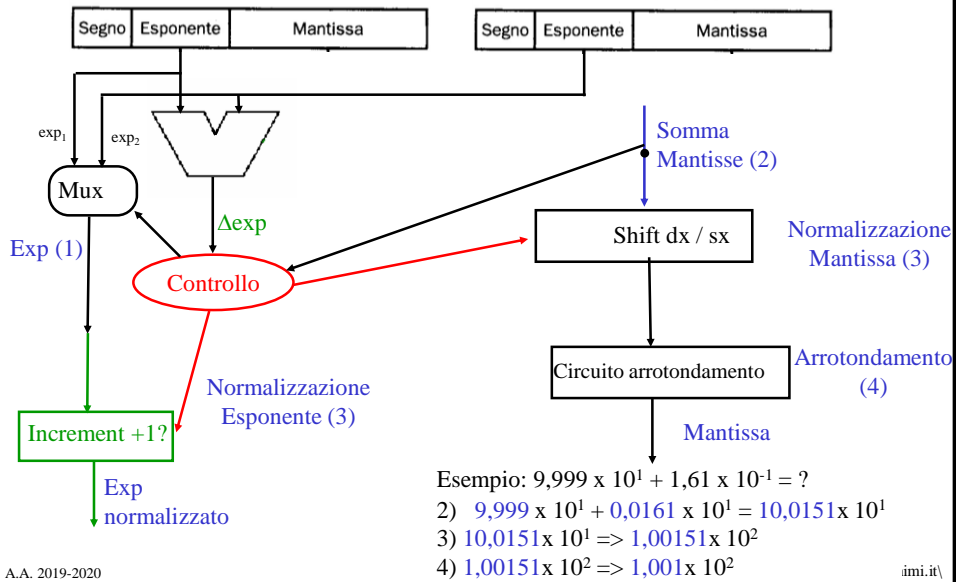
## Il circuito della somma floating point: allineamento delle mantisse e somma







# Il circuito della somma floating point: arrotondamento e normalizzazione



# Circuito della somma floating point

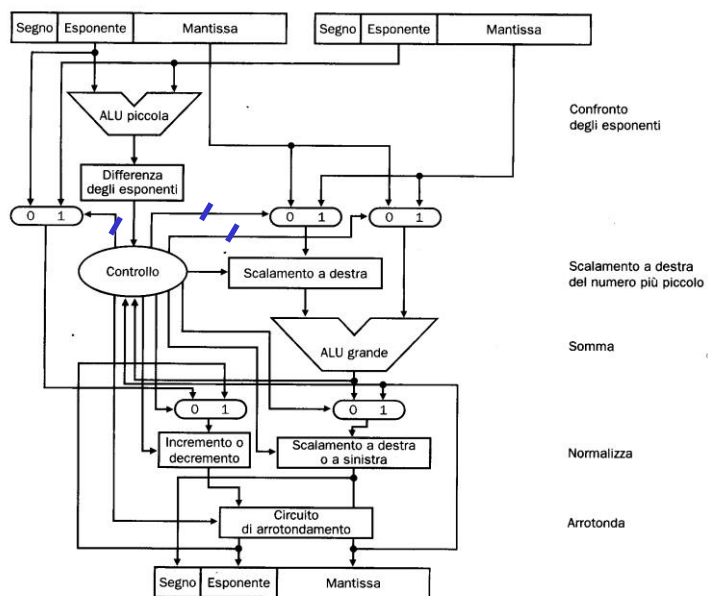


Le tre linee in blu contengono lo stesso segnale di controllo.

Gestisce anche la rinormalizzazione:  
 $9,99999 \times 10^2 = 10,00 \times 10^3$

Gestisce anche i numeri negativi.

Problemi?





# Problemi di troncamento



$$A = 4$$

$$B = 1,0000003576278686523438 \times 10^0$$

In IEEE754:

$$A = 1 \times 2^2$$

$$B = 1,000000000000000000000011 \times 2^0$$

$$A = 0 \ 1000 \ 0001 \ 00000 \ 00000 \ 00000 \ 00000 \ 000$$

$$B = 0 \ 1111 \ 1111 \ 00000 \ 00000 \ 00000 \ 00000 \ 011$$

$$\text{Allinea B ad A} \Rightarrow B = 0,01000 \ 00000 \ 00000 \ 00000 \ 00011 \times 2^2$$

Segue che:

$$C = A + B = 1,01000 \ 00000 \ 00000 \ 00000 \ 00011 \times 2^2$$

$$\text{Codificando } C = 0 \ 1000 \ 0001 \ 01000 \ 00000 \ 00000 \ 00000 \ 000 = 5d_{cc}$$



# Circuito della somma floating point con bit di arrotondamento

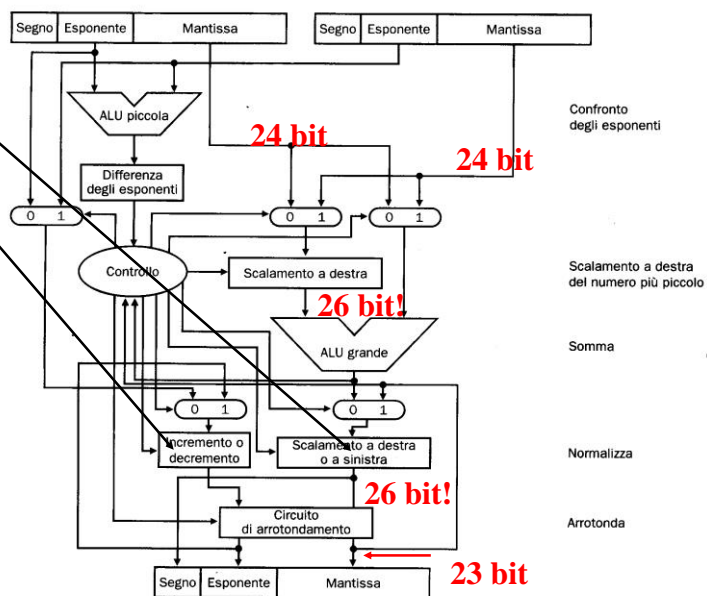


In quale caso la mantissa viene scalata a sx?

In quale caso l'esponente viene decrementato?

La rappresentazione interna, secondo IEEE 754, prevede 2 bit aggiuntivi: **bit di guardia** e **bit di arrotondamento**.

Mantissa 1,...





## Prodotto e divisione in virgola mobile



- Prodotto delle mantisse
- Somma degli esponenti
- Normalizzazione
  
- Divisione in virgola mobile = Prodotto di un numero per il suo inverso.



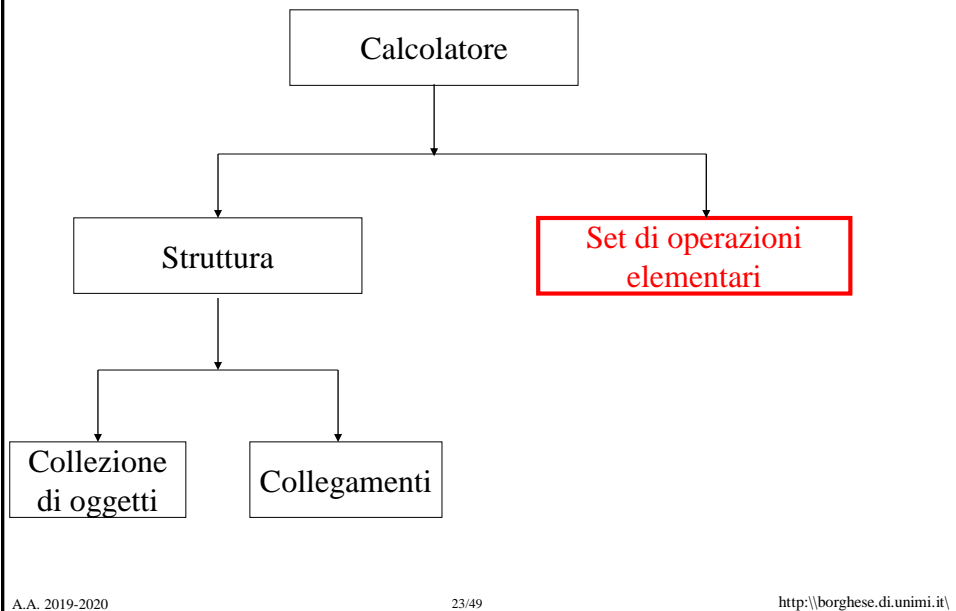
## Sommario



- Somma in virgola mobile
- **ISA**
- Istruzioni aritmetico-logiche



## Descrizione di un elaboratore



## Definizione di un'ISA



*Definizione del funzionamento: insieme delle istruzioni (interfaccia verso i linguaggi ad alto livello).*

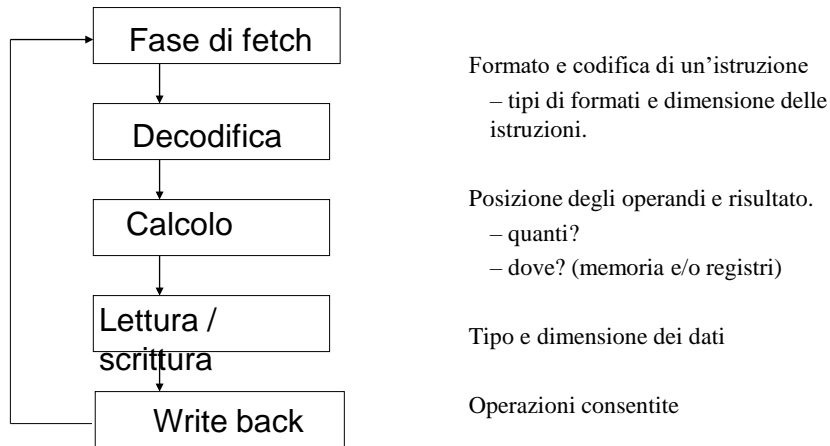
- Tipologia di istruzioni.
- Meccanismo di funzionamento.

*Definizione del formato: codifica delle istruzioni (interfaccia verso l'HW).*

- Formato delle istruzioni.
- Suddivisione in gruppi omogenei dei bit che costituiscono l'istruzione.



## Caratteristiche di un'ISA



## Le istruzioni di un'ISA

Devono contenere tutte le informazioni necessarie ad eseguire il ciclo di esecuzione dell'istruzione: registri, comandi, ....

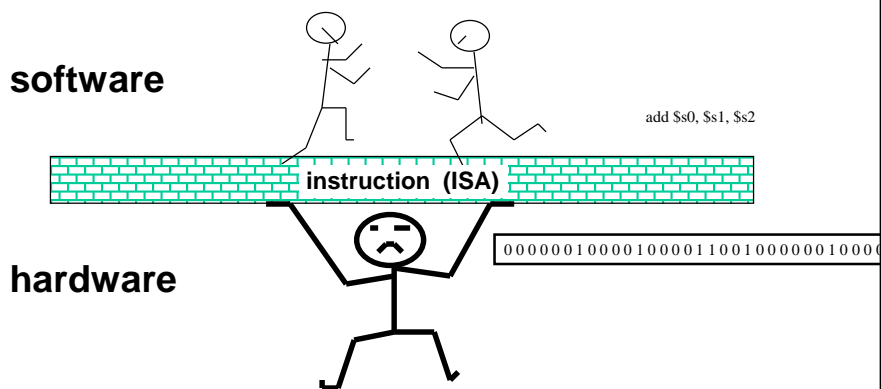
### Ogni architettura di processore ha il suo linguaggio macchina

- Architettura dell'insieme delle istruzioni elementari messe a disposizione dalla macchina (in linguaggio macchina).
  - **ISA (Instruction Set Architecture)**
- Due processori con lo stesso linguaggio macchina hanno la stessa architettura delle istruzioni anche se le implementazioni hardware possono essere diverse.
- Consente al SW di accedere direttamente all'hardware di un calcolatore.

*L'architettura delle istruzioni, specifica come vengono costruite le istruzioni in modo tale che siano comprensibili alla macchina (in linguaggio macchina).*



## Insieme delle istruzioni



Quale è più facile modificare?



## Tipi di istruzioni

- Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:
  - Istruzioni aritmetico-logiche;
  - Istruzioni di trasferimento da/verso la memoria (*load/store*);
  - Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;
  - Istruzioni di trasferimento in ingresso/uscita (I/O).



## Le istruzioni in linguaggio macchina

- Linguaggio di programmazione direttamente comprensibile dalla macchina
  - Le parole di memoria sono interpretate come *istruzioni*
  - Vocabolario è *l'insieme delle istruzioni (instruction set)*

Programma in  
linguaggio ad alto livello  
(C)

```
a = a + c  
b = b + a  
var = m [a]
```



Programma in linguaggio  
macchina

```
011100010101010  
000110101000111  
000010000010000  
001000100010000
```



## Linguaggio Assembler

- Le istruzioni assembler sono una rappresentazione simbolica del linguaggio macchina comprensibile dall'HW.
- Rappresentazione simbolica del linguaggio macchina
  - Più comprensibile del linguaggio macchina in quanto utilizza simboli invece che sequenze di bit
- Rispetto ai linguaggi ad alto livello, l'assembler fornisce limitate forme di controllo del flusso e non prevede articolate strutture dati
- Linguaggio usato come linguaggio target nella fase di compilazione di un programma scritto in un linguaggio ad alto livello (es: C, Pascal, ecc.)
- Vero e proprio linguaggio di programmazione che fornisce la visibilità diretta sull'hardware.



## Linguaggio C: somma dei primi 100 numeri al quadrato



```
main()
{
    int i;
    int sum = 0;
    for (i = 0; i <= 100; i = i + 1)
        sum = sum + i*i;
    printf("La somma da 0 a 100 è
%d\n", sum);
}
```



## Linguaggio Assembler: somma dei primi 100 numeri al quadrato



```
.text
.align 2
.globl main
main:
    add $t6, $zero, $zero
    add $s0, $zero, $zero
    add $s1, $a0, $zero
loop: mult $t4, $t6, $t6
    addu $s0, $s0, $t4
    addu $t6, $t6, 1
    bne $t6, $s1, loop
.....
```





## Assembler come linguaggio di programmazione



- Principali *svantaggi* della programmazione in linguaggio assembler:
  - Mancanza di portabilità dei programmi su macchine diverse
  - Maggiore lunghezza, difficoltà di comprensione, facilità d'errore rispetto ai programmi scritti in un linguaggio ad alto livello
  
- Principali *vantaggi* della programmazione in linguaggio assembler:
  - Ottimizzazione delle prestazioni.
  - Massimo sfruttamento delle potenzialità dell'hardware sottostante.
  
- Le strutture di controllo hanno forme limitate
- Non esistono tipi di dati all'infuori di interi, virgola mobile e caratteri.
- La gestione delle strutture dati e delle chiamate a procedura deve essere fatta in modo esplicito dal programmatore.



## Assembler come linguaggio di programmazione



- Alcune applicazioni richiedono un approccio *ibrido* nel quale le parti più critiche del programma sono scritte in assembly (per massimizzare le prestazioni) e le altre parti sono scritte in un linguaggio ad alto livello (le prestazioni dipendono dalle capacità di ottimizzazione del compilatore).

Esempio: Sistemi embedded o dedicati

Sistemi “automatici” di traduzione da linguaggio ad alto livello (linguaggio C) ad Assembly e codice binario ed implementazione circuitale (e.g. sistemi di sviluppo per FPGA).



## I registri



- Un registro è un insieme di celle di memoria che vengono lette / scritte in parallelo.
- I registri sono associati alle variabili di un programma dal compilatore. Contengono i **dati**.
- Un processore possiede un numero limitato di registri: ad esempio il processore MIPS possiede **32 registri composti da 32-bit (word), register file**.
- I registri possono essere direttamente indirizzati mediante il loro numero progressivo (0, ..., 31) preceduto da \$: ad es.  
**\$0, \$1, ..., \$31**
- Per convenzione di utilizzo, sono stati introdotti nomi simbolici significativi. Sono preceduti da \$, ad esempio:

**\$s0, \$s1, ..., \$s7 (\$s8)** Per indicare variabili in C

**\$t0, \$t1, ... \$t9** Per indicare variabili temporanee



## I registri del register file



	Nome	Numero	Utilizzo
→	\$zero	0	costante zero
	\$at	1	riservato per l'assemblatore
	\$v0-\$v1	2-3	valori di ritorno di una procedura
	\$a0-\$a3	4-7	argomenti di una procedura
→	\$t0-\$t7	8-15	registri temporanei (non salvati)
→	\$s0-\$s7	16-23	registri salvati
→	\$t8-\$t9	24-25	registri temporanei (non salvati)
	\$k0-\$k1	26-27	gestione delle eccezioni
	\$gp	28	puntatore alla global area (dati)
	\$sp	29	stack pointer
	\$s8	30	registro salvato (fp)
	\$ra	31	indirizzo di ritorno



## I registri per le operazioni floating point

- Esistono 32 registri utilizzati per l'esecuzione delle istruzioni.
- Esistono **32** registri per le operazioni floating point (virgola mobile) indicati come

**\$f0, ..., \$f31**

- Per le operazioni in doppia precisione si usano i registri contigui

**\$f0, \$f2, \$f4, ...**



## Sommario

- Somma in virgola mobile
- ISA
- Istruzioni aritmetico-logiche



## Tipi di istruzioni



```
for (i=0; i<N; i++)           // Istruzioni di controllo
{ elem = i*N + j;             // Istruzioni aritmetico-logiche
  s = v[elem];                 // Istruzioni di accesso a memoria
  z[elem] = s;                 // Istruzioni di accesso a memoria
}
```



## Tipi di istruzioni



- Le istruzioni comprese nel linguaggio macchina di ogni calcolatore possono essere classificate nelle seguenti quattro categorie:
  - Istruzioni aritmetico-logiche;
  - Istruzioni di trasferimento da/verso la memoria (*load/store*);
  - Istruzioni di salto condizionato e non condizionato per il controllo del flusso di programma;
  - Istruzioni di trasferimento in ingresso/uscita (I/O).



## Istruzioni aritmetico-logiche



- In MIPS, un'istruzione aritmetico-logica possiede in generale *tre* operandi: i due registri contenenti i valori da elaborare (*registri sorgente*) e il registro contenente il risultato (*registro destinazione*).
- L'ordine degli operandi è **fisso**: prima il registro contenente il risultato dell'operazione e poi i due operandi.
- L'istruzione assembly contiene il codice operativo e tre campi relativi ai tre operandi:

```
OPCODE DEST, SORG1, SORG2
```

Le operazioni vengono eseguite esclusivamente su dati presenti nella CPU, non su dati residenti nella memoria.



## Esempi: istruzioni add e sub



Codice C:

```
R = A + B;
```

Codice assembler MIPS:

```
add $s16, $s17, $s18  
add rd, rs, rt
```

mette la somma del contenuto di rs e rt in rd:

```
add rd, rs, rt      # rd ← rs + rt
```

Nella traduzione da linguaggio ad alto livello a linguaggio assembly, le variabili sono associate ai registri dal compilatore

**sub** serve per sottrarre il contenuto di due registri sorgente rs e rt:

```
sub rd rs rt
```

e mettere la differenza del contenuto di rs e rt in rd

```
sub rd, rs, rt      # rd ← rs - rt
```



## Istruzioni aritmetico-logiche in sequenza



Il fatto che ogni istruzione aritmetica ha tre operandi sempre nella stessa posizione consente di semplificare l'hw, ma complica alcune cose...

Codice C:  $Z = A - (B + C + D) \Rightarrow$   
 $E = B + C + D; Z = A - E;$

Suppongo che le variabili siano contenute nei seguenti registri:

A -> \$s0    B -> \$s1    C -> \$s2    D -> \$s3    Z -> \$s5

Codice MIPS:

```
add $t0, $s1, $s2
add $t1, $t0, $s3
sub $s5, $s0, $t1
```



## Istruzioni aritmetico-logiche



- Operazioni con un numero di operandi maggiore di tre possono essere effettuate scomponendole in operazioni più semplici.
- Ad esempio, per eseguire la somma e sottrazione delle variabili A . . D nella variabile Z servono tre istruzioni :

Codice C:  $Z = A + B - C + D$

Codice MIPS:

```
add $t0, $s0, $s1
sub $t1, $t0, $s2
add $s5, $t1, $s3
```



## Implementazione alternativa



- Operazioni con un numero di operandi maggiore di tre possono essere effettuate scomponendole in operazioni più semplici.
- Ad esempio, per eseguire la somma e sottrazione delle variabili A . . D nella variabile Z servono tre istruzioni :

Codice C:  $Z = (A + B) - (C - D)$

Codice MIPS:  
`add $t0, $s0, $s1`  
`sub $t1, $s2, $s3`  
`sub $s5, $t0, $t1`

Quale implementazione è la migliore? Scegliere il compilatore il quale cerca di massimizzare la parallelizzazione del codice.



## Moltiplicazione



- Due istruzioni:
  - `mult rs rt`
  - `multu rs rt` `# unsigned`
- Il registro destinazione è *implicito*.
- Il risultato della moltiplicazione viene posto sempre in due registri dedicati di una parola (special purpose) denominati *hi* (*High order word*) e *lo* (*Low order word*)
- La moltiplicazione di due numeri rappresentabili con 32 bit può dare come risultato un numero non rappresentabile in 32 bit



## Moltiplicazione



- Il risultato della moltiplicazione si preleva dal registro **hi** e dal registro **lo** utilizzando le due istruzioni:

- **mfhi rd** # move from hi
  - Sposta il contenuto del registro **hi** nel registro **rd**
- **mflo rd** # move from lo
  - Sposta il contenuto del registro **lo** nel registro **rd**

Test sull'overflow

Risultato del prodotto



## add: varianti



- **addi \$s1, \$s2, 100** #add immediate
  - Somma una costante: il valore del secondo operando è presente nell'istruzione come costante e sommata estesa in segno.  
 $rt \leftarrow rs + \text{costante}$
- **addiu \$s0, \$s1, 100** #add immediate unsigned
  - Somma una costante ed evita overflow.
- **addu \$s0, \$s1, \$s2** #add unsigned
  - Evita overflow: la somma viene eseguita considerando gli addendi sempre positivi. Il bit più significativo è parte del numero e non è bit di segno.

Non esiste un'istruzione di subi. Perché?





## Sommario



- Somma in virgola mobile
- ISA
- Istruzioni aritmetico-logiche